Python

1

Переменные

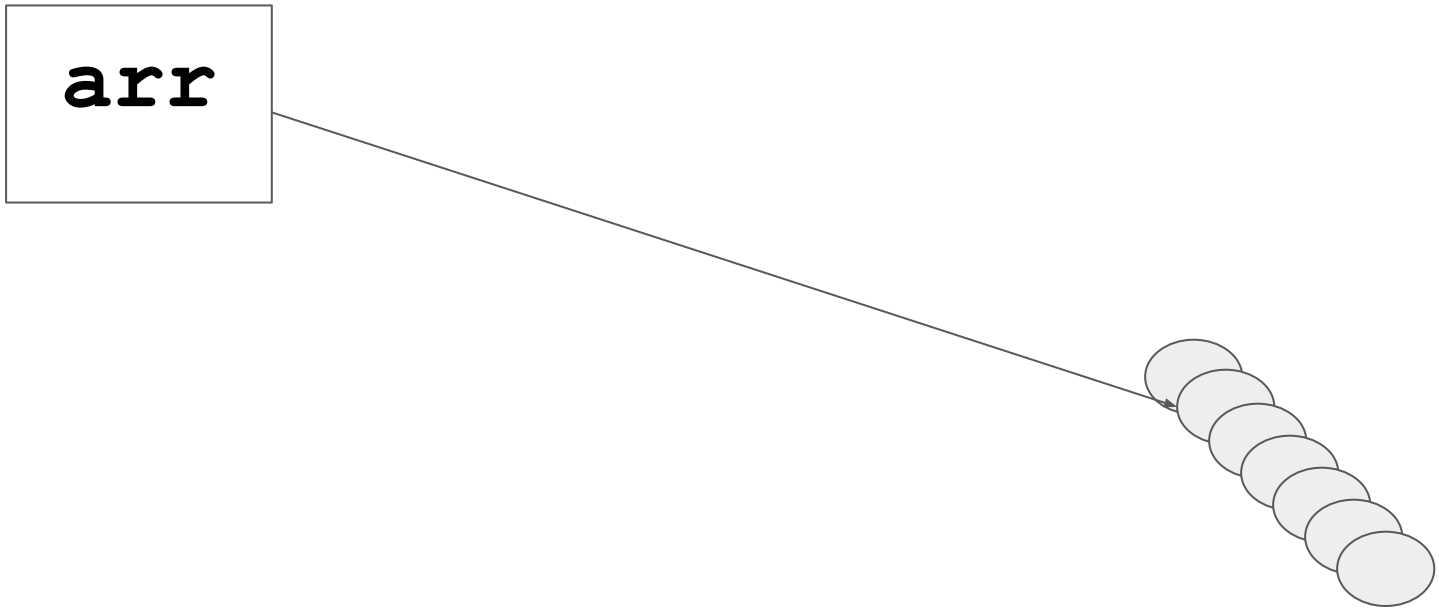
Переменная — это простейшая именованная структура данных, в которой может быть сохранён промежуточный или конечный результат работы программы.

Переменную в Python создать очень просто — нужно присвоить некоторому идентификатору значение при помощи оператора присваивания «=».

```
a = 10
```

Никакого специального объявления переменных не требуется, первое присваивание переменной значения и является ее объявлением. Идентификатор в Python является "ссылкой" на хранимые в памяти данные.

```
arr = [1, 2, 3]
```



Типы данных

В языке Python выделяют несколько типов данных: целые числа, числа с плавающей точкой(вещественные), строки, логический тип, списки, множества, кортежи, словари.

Целые числа

Целое число в Python имеет тип **int**. Оно записывается как последовательность цифр, перед которой также может стоять знак минус. Является неизменяемым типом

10, 30, -1, -10000000, 24252352365467586978965746535 4

- *ОСНОВНЫЕ ОПЕРАЦИИ С ЦЕЛЫМИ ЧИСЛАМИ*
- *A + B — сумма;*
- *A - B — разность;*
- A % B — взятие остатка от деления A на B;
- A // B — взятие целой части от деления A на B
- A ** B — возведение в степень.

Приоритет операций

Приоритеты операций в Python совпадают с приоритетом операций в математике, а именно:

1. Выполняются возведения в степень справа налево, то есть 3 ** 3 ** 3 это 3 ** (3 ** 3).
2. Выполняются унарные минусы (отрицания).
3. Выполняются умножения и деления слева направо. Операции умножения и деления имеют одинаковый приоритет.
4. Выполняются сложения и вычитания слева направо. Операции сложения и вычитания имеют одинаковый приоритет.

Для изменения порядка действий нужно использовать скобки.

Операторы и операнды

+ сложение	Складывает значение переменных слева и справа от оператора	a + b = 30
- вычитание	Вычитает из левого операнда правый	a – b = -10
* умножение	Перемножает левый и правый операнды	a * b = 200
/ деление	Делит левый операнд на правый	b / a = 2
% остаток от деления	Делит левый операнд на правый и возвращает остаток от деления	b % a = 0 и 42%10 = 2
** возведение в степень	Возводит левый операнд в степень равную правому оператору	5**2 =25 и 2**3 = 8
// целочисленное деление	Делит левый операнд на правый и оставляет только целую часть.	9//2 = 4 and 9.0//2.0 = 4.0, -11//3 = -4, -11.0//3 = -4.0

Числа с плавающей точкой (вещественные)

Вещественное число в Python имеет тип float. Оно записывается как последовательность цифр, перед которой также может стоять знак минус. В качестве разделителя целой и дробной части используется точка. Является неизменяемым типом

2.4, 5.0, 2323.000

- A + B — сумма;
- A - B — разность;
- A * B — произведение;
- A / B — частное, *(результатом этого действия является вещественное число, даже если A нацело делится на B);*
- A % B — взятие остатка от деления A на B, *(подразумевается, что неполное частное является целым числом);*
- A // B — взятие целой части от деления A на B, *(подразумевается, что неполное частное является целым числом);*
- A ** B — возведение в степень.

Обновление переменных (x=x+1)

1) переменные могут использованы для установки значений других переменных

```
my_age = 17
alisa_age = my_age + 4
fred_age = alisa_age + 1
```

2) "обновление самой себя"

```
my_age = 10
my_age = my_age + 5
my_age = my_age + 1
my_age += 1
```

Строки

Строки в Python имеют тип str. Строкой называется последовательность символов: букв, цифр, знакоы препинания и т.д. Является неизменяе

“ ”, ‘ Some string’

Основные операции со строками

- A + B — конкатенация (строка B приписывается к строке A);
- A * n — повторение n раз, значение n должно быть целого типа.

Операции со строками

Сложение строк(Конкатенация).

>>> a = ‘Hello’

>>> b = ‘World’

>>> a + ‘, ’ + b

Hello, World

Определение длины строки

>>> len(a)

5

Обращение по индексу

>>> a[0]

H

>>> a[1:3]

el

Некоторые функции работы со строками

```
>>> st = ‘Hello world’

>>> st.lower()

hello world

>>> st.isdigit()

False

>>> ‘42’.isdigit()

True
```

```
>>> st.find(‘l’)

2

>>> st.rfind(‘l’)

9

>>> st.find(‘a’)

-1
```

Списки

Список представляет собой упорядоченную последовательность элементов. Он очень гибкий и является одним из самых используемых типов в Python. Элементы списка не обязательно должны быть одного типа.

```
a = [1, 2.2, 'python']
```

Индексация

Мы можем использовать оператор `[]` для извлечения элемента (такая операция называется “доступ по индексу”) или диапазона элементов (такая операция называется “извлечение среза”) из списка. В Python индексация начинается с нуля:

```
>>> a = [5,10,15,20,25,30,35,40]
>>> print("a[2] =", a[2]) a[2] = 15
>>> print("a[0:3] =", a[0:3]) a[0:3] = [5, 10, 15]
>>> print("a[5:] =", a[5:]) a[5:] = [30, 35, 40]
```

Списки являются изменяемым типом, т.е. значения его элементов можно изменить:

```
>>> a = [1,2,3]
>>> a[2] = 4
>>> a [1, 2, 4]
```

Операции со списками

>>> list_a = [‘a’, ‘b’, 42]

Добавление

>>> list_a.append(‘p’)

Обращение к элементу списка

>>> list_a[0]

a

Удаление элемента по значению

>>> list_b.remove(‘a’)

Удаление по индексу

>>> del list_b[0]

Изменение элемента списка

>>> list_a[0] = 77

Некоторые методы работы со списками

Расширение

```
>>> a = [1, 2, 3]
>>> b = [7, 8, 9]
>>> b.extend(a)
>>> print(b)
[7, 8, 9, 1, 2, 3]
```

Длина списка

```
>>> len(a)
```

Вставление элемента

```
>>> a.insert(0, 5)
>>> print(a)
[5, 1, 2, 3]
```

Удаление элемента по индексу с возвращением значения

```
>>>a.pop(2)
3
```

Кортежи (tuple)

Так же как и список, кортеж (tuple) является упорядоченной последовательностью элементов. Вся разница заключается в том, что кортежи неизменяемы.

Кортежи используются для защиты данных от перезаписи и обычно работают быстрее, чем списки, т.к. их нельзя изменять.

Для создания кортежа нужно поместить внутрь круглых скобок элементы, разделённые запятой:

```
>>> t = (5,'program', 1+3j)
```

Срезы

Мы можем использовать оператор извлечения среза `[]` для извлечения элементов, но мы не можем менять их значения:

```
>>> t = (5,'program', 1+3j)
>>> print("t[1] =", t[1])
t[1] = program
>>> print("t[0:3] =", t[0:3])
t[0:3] = (5, 'program', (1+3j))
# Приводит к ошибке, т.к.
# кортежи неизменяемы
>>> t[0] = 10
```

Множества

Множество является неупорядоченной уникализированной последовательностью. Объявляется множество с помощью элементов, разделённых запятой, внутри фигурных скобок:

```
>>> a = {5,2,3,1,4}
# вывод переменной множества
>>> print("a =", a)
a = {1, 2, 3, 4, 5}
# тип данных переменной a
>>> print(type(a))
<class 'set'>
```

Над множествами можно выполнять такие операции, как объединение и пересечение. Т.к. элементы в множестве должны быть уникальны, они автоматически удаляют дубликаты:

```
>>> a = {1,2,2,3,3,3}
>>> a
{1, 2, 3}
```

Поскольку множество является неупорядоченной последовательностью, оператор извлечения среза здесь не работает:

```
>>> a = {1,2,3}
>>> a[1]
Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
    TypeError: 'set' object does not support indexing
```

Словари

Словари — неупорядоченные наборы пар ключ-значение. Имеют свойство изменяться.

Они используются, когда нужно сопоставить каждому из ключей значение и иметь возможность быстро получать доступ к значению, зная ключ. В других языках словари обычно называются `map`, `hash` или `object`. Словари оптимизированы для извлечения данных. Чтобы извлечь значение, нужно знать ключ.

Словари

Словарь объявляется парами элементов в форме ключ:значение, заключенными в фигурные скобки:

```
>>> d = {1:'value', 'key':2}

>>> type(d)

 <class 'dict'>
```

*

Значение может быть любого типа, а вот ключ — только неизменяемого.

Мы используем ключ, чтобы получить соответствующее ему значение. Но не наоборот:

```
>>> d = {1:'value', 'key':2}
>>> print("d[1] =", d[1]);
d[1] = value
>>> print("d['key'] =", d['key']);
d['key'] = 2
# Приводит к ошибке
>>> print("d[2] =", d[2]);
```

Преобразование типов

Мы можем преобразовывать значения из одного типа в другой с помощью таких функций, как `int()`, `float()`, `str()` и т.д.

```
>>> float(5)
5.0
```

При преобразовании числа с плавающей запятой в целое будет утеряна часть после запятой:

```
>>> int(10.6)
10
>>> int(-10.6)
-10
```

Для преобразования из/в строку должны использоваться совместимые значения

```
>>> float('2.5')
2.5
>>> str(25)
'25'

>>> int('1p')
Traceback (most recent call last):
   File "<stdin>", line 1, in <module>
ValueError: invalid literal for int() with base 10: '1p'
```

Преобразование последовательностей

Можно даже преобразовывать одну последовательность в другую:

```
>>> set([1,2,3])

{1, 2, 3}

>>> tuple({5,6,7})

(5, 6, 7)

>>> list('hello')

 ['h', 'e', 'l', 'l', 'o']
```

list -> str

Для преобразования списка из символов обратно в строку нельзя вызвать `str(список)`, так как в результате мы получим строковое представление списка (наподобие того, что мы видим, когда выводим список на экран). Вместо этого нужно сделать следующее:

```
''.join(['h', 'e', 'l', 'l', 'o'])
```

Для преобразования в словарь каждый элемент последовательности должен быть парой:

```
>>> dict([[1,2],[3,4]])

{1: 2, 3: 4}

>>> dict([(3,26),(4,44)])

 {3: 26, 4: 44}
```